Технология программирования

Лекция 1

Зариковская Наталья Вячеславовна,

доцент кафедры ЭМИС

Список рекомендуемой литературы

- 1. Дж. Фокс. Программное обеспечение и его разработка. М.: ≪Мир≫, 1982
- 2. И. Соммервиль. Инженерия программного обеспечения. – М.: Изд. ≪Вильямс≫, 2002.
- 3. Ф. Брукс. Мифический человеко-месяц или Как создаются программные системы. – СПб.: Изд. ≪Символ-плюс≫, 2000.
- 4. Э. Дейкстра. Дисциплина программирования. – М.: 1982.
- 5. М. Кантор. Управление программными проектами. Практическое руководство по разработке успешного программного обеспечения – М.: ≪Вильямс≫, 2002.
- 6. В.В. Липаев. Тестирование программ. – М.: ≪Радио и связь≫, 1986, 296 с.
- 7. Сайт ≪Microsoft Solutions Framework. Методология создания программных решений≫, http://www.microsoft.com/rus/msdn/msf/default.mspx – Книга ≪Анализ требований и создание архитектуры решений на основе Microsoft .NET. Учебный курс MCSD≫, Русская редакция, 2004, 416 с.

8. ГОСТ 19.001-77. Единая система программной документации. Общие положения.

Список рекомендуемой литературы

- 9. сайт http://www.fsf.org/
- 10. А.Н. Терехов, Л.А. Эрлих, А.А. Терехов. История и архитектура проекта RescueWare. Автоматизированный реинжиниринг программ. Под ред. А.Н. Терехова, А.А. Терехова– СПб.: Изд-во С.-Петербургского университета, 2000, с. 7-19.
- 11. М.А. Бульонков, Д.Е. Бабурин, HyperCode – открытая система визуализации программ. Автоматизированный реинжиниринг программ. Под ред. А.Н. Терехова, А.А. Терехова– СПб.: Изд. С.-Петербургского университета, 2000, с. 165-183.
- 12. Р. Фатрелл, Д. Шафер, Л. Шафер. Управление программными проектами: достижение оптимального качества при минимуме затрат – М., ≪Вильямс≫, 2003.
- 13. Международный стандарт ISO 9001-94. Системы качества. Модель обеспечения качества при проектировании, разработке, производстве, монтаже и обслуживании – М.: ИПК, Изд. стандартов, 1996, 19 с.
- 14. В.Е. Карпов, К.А. Коньков. Основы операционных систем. Курс лекций. Учебное пособие. – М.: Интеренет-Университет Информационных Технологий, 2004, 628 с.
- 15. А.Н. Терехов, К.Ю. Романовский, Дм.В. Кознов, П.С. Долгов, А.Н. Иванов. REAL: методология и CASE-средство для разработки систем реального времени и информационных систем//Программирование, 1999, № 5. с. 44-52.

Часть I. Общие вопросы технологии программирования

- **1. Понятие технологии программирования**
- **Технология** – это набор правил, методик и инструментов, позволяющих наладить ***производственный*** процесс выпуска какого-либо продукта.
- Отнесем сюда: процессы планирования, измерения, оценки качества, ответственность исполнителя и многое другое.

- **2. Жизненный цикл программы**
- Программный продукт является результатом некоего производственного процесса.
- Этот процесс нужно спланировать, оценить ресурсы, для чего, в свою очередь, требуются более или менее точные спецификации, что же необходимо заказчику. Затем продукт надо спроектировать в виде системы, состоящей из многих компонентов, описать функции этих компонентов и их связи между собой, после чего компоненты нужно запрограммировать, автономно отладить, собрать вместе, провести комплексную отладку, подготовить документацию на систему, обучить пользователей, провести опытную эксплуатацию и организовать сопровождение системы на весь период ее эксплуатации.
- ≪жизненный цикл программы≫

- Почему ≪цикл≫?
- Потому что редко разработка развивается столь прямолинейно, хотя одну из первых моделей ЖЦП действительно назвали ≪водопадная≫, подчеркивая тот факт, что к предыдущей фазе проектирования вернуться невозможно. Действуя по этой модели, коллектив последовательно разрабатывает проект – от исходной концепции до комплексного тестирования. Модель требует определить опорные точки, в которых будет оцениваться сделанное и решаться вопрос о том, можно ли двигаться дальше.
- Такой подход хорош для проектов, в которых требования легко формулируются с самого начала, но не годится для сложных, когда требования могут неоднократно меняться. Кроме того, водопадная модель вынуждает готовить огромную массу документации и требует единообразной процедуры оценки результатов на каждом этапе. Эти две особенности часто приводят к синдрому ≪аналитического паралича≫, напряженным отношениям между разработчиками, заказчиками и пользователями.
- В реальной жизни, конечно же, чисто водопадная модель не применяется. Появляются новые требования заказчиков, изменяется аппаратура, находятся такие ошибки, которые невозможно исправить, не затронув результаты предыдущих фаз и т.д.

- Развитие системы – это именно циклическое повторение практически всех фаз, постоянный возврат к предыдущим фазам. Если не принять специальных мер (что, собственно, и является предметом технологии программирования), процесс может стать бесконечным.
- Одной из первых практически полезных моделей ЖЦП стала модель создания прототипов. С самого начала разработчики пытаются выделить основные, существенные требования заказчика и реализовать только их в виде работающего прототипа системы. Этот прототип демонстрируется заказчику. Часто бывает, что заказчик в ужасе кричит, что его неправильно поняли, он хотел совсем другого, зато теперь он хоть может внятно сформулировать свои требования, глядя на работу прототипа. Цикл разработки и показа прототипа повторяется несколько раз, пока заказчик не скажет: ≪Да, это, кажется, то, что мне нужно≫. Только после этого дорабатываются куски, выброшенные в начале разработки, подготавливается документация, короче, делаются многие вещи, на которые время было бы потрачено зря, если бы их делали для самого первого, неудачного прототипа.
- Для небольших систем, особенно для тех, в которых велик процент интерактивных (взаимодействующих с пользователем) компонентов, такая модель работает.
- Однако, существует множество случаев, когда важные аспекты просто упускались при разработке прототипа, из-за чего позже все приходилось переделывать заново.

- Некоторым обобщением модели создания прототипов является спиральная модель, в которой разработка приложения выглядит как серия последовательных итераций. На первых этапах уточняются спецификации продукта, на последующих — добавляются новые возможности и функции.
- Цель этой модели – по окончании каждой итерации осуществить заново оценку рисков продолжения работ. Программисты часто увлекаются технической сутью выполняемого проекта и не видят общей картины, особенно в части производственных затрат. Нам все кажется, что вот еще немного, еще чуть-чуть, и все проблемы будут решены, но ≪асфальтовая топь≫ (по выражению Фредерика Брукса) засасывает нас, не давая шансов достичь твердых осязаемых результатов.
- Один бизнесмен представил эту проблему так: ≪Вот ты затратил 100 000 долларов, но задачу пока не решил. Нужно сто раз подумать, что лучше — истратить еще столько же, чтобы успешно завершить проект, или через год снова оказаться перед тем же выбором, но тогда уже с риском потерять не 100 000, а 200 000 долларов?≫
- В силу своей итеративной природы спиральная модель допускает корректировки по ходу работы, что способствует улучшению продукта.
- При большом числе итераций разработка по этой модели нуждается в глубокой автоматизации всех процессов, иначе она становится неэффективной. На практике у заказчиков и пользователей иногда возникает ощущение нестабильности продукта, так как они не успевают уследить за слишком быстрыми изменениями в нем.

- Один очень важный вывод мы можем сделать даже при таком начальном знакомстве с понятием ЖЦП. Собственно программирование не является единственным занятием коллектива, занятого промышленными разработками. Более того, оно не является даже главным, наиболее трудоемким делом. Многие исследования отдают на фазу программирования не более 15-20% времени, затраченного на разработку (сопровождение вообще бесконечно). Может быть, эти цифры заставят вас задуматься о важности и других аспектов образования – от умения найти и обосновать эффективный алгоритм до искусства владения родным языком, как устным, так и письменным.

3. Постановка задачи. Оценка осуществимости

- Обычно заказчик выдает две-три страницы текста задания и сразу же просит оценить время исполнения заказа и его стоимость. Надо быть сумасшедшим, чтобы на это согласиться. Нередки случаи, когда целые коллективы ошибаются в пять-десять раз и попадают в кабалу или теряют профессиональную репутацию. Чтобы избежать такой ситуации, нужно предложить заказчику оформить начальный договор на две-четыре недели с тем чтобы два-три системных аналитика разобрались в задаче, с помощью каких-то инструментальных средств выполнили декомпозицию системы на компоненты, прикинули возможные объемы этих компонентов и, соответственно, время их реализации. Такая начальная стадия ЖЦП называется ≪**оценкой осуществимости**≫.

- Можно, конечно, выполнить эту работу за свой счет (и многие крупные предприятия так и делают), но, во-первых, отношение к внутренним разработкам – более спокойное, а, во-вторых, оплаченный договор гарантирует серьезность намерений обеих сторон. К сожалению, часто бывает, что недобросовестные заказчики пользуются таким приемом, чтобы бесплатно получить идеи разработки или просто выполнить экспертизу оценок своих специалистов без всякого намерения передать разработку на сторону.

3. Постановка задачи. Оценка осуществимости

- Постановка задачи – наиболее творческая часть ЖЦП, которая поднимает почти философские проблемы.
- Требуется описать поведение разрабатываемой системы. Эта система получает какие-то сигналы из ее окружения, поэтому надо описать поведение окружения, но окружение само зависит и изменяется под влиянием системы, ее сигналов, особенно аварийных.
- Разрешают это противоречие, постепенно уточняя поведение как системы, так и ее окружения. Для действительно важных систем заказчик требует разработки имитационных моделей системы и окружения, не уступающих по сложности и детальности самой системе.

3. Постановка задачи. Оценка осуществимости

- Несколько слов о декомпозиции системы. Следует помнить, что оценка сложности системы, которая является суммой оценок ее компонентов, полученных в результате декомпозиции, намного точнее, чем первоначальная оценка системы в целом. Ведь делают эти оценки одни и те же системные аналитики, наверняка у них уже было примерное представление о системе и ее компонентах, когда они давали первоначальную оценку. И тем не менее, эти же мысли, положенные на бумагу, особенно с помощью каких-то средств формализации, приводят к совершенно другому результату.
- Однако, формализация формализации рознь. Ранее бытовало мнение, что можно и нужно каждую задачу формально специфицировать с помощью логики предикатов с тем чтобы можно было доказать корректность ее решения. Для каждой программы P нужно задать предусловие A, описывающее состояние среды (в частности, переменных программы) перед исполнением программы, и постусловие S, описывающее состояние после завершения ее исполнения:
- A {P} S
- Очевидно, что A и S – несопоставимы, поскольку описывают состояние в разные моменты времени. Можно ≪протянуть≫ предусловие A через текст программы, изменяя его соответственно каждому проходимому оператору, при этом логическая формула предусловия фантастически быстро растет, например, после условного предложения будет дизъюнкция двух вариантов, описывающих текущие предусловия после веток then и else – мы же не знаем, какая именно ветка будет исполняться.

Обозначим результат протягивания через A'. Тогда остается доказать истинность импликации A'⇒S, и программа корректна.

3. Постановка задачи. Оценка осуществимости

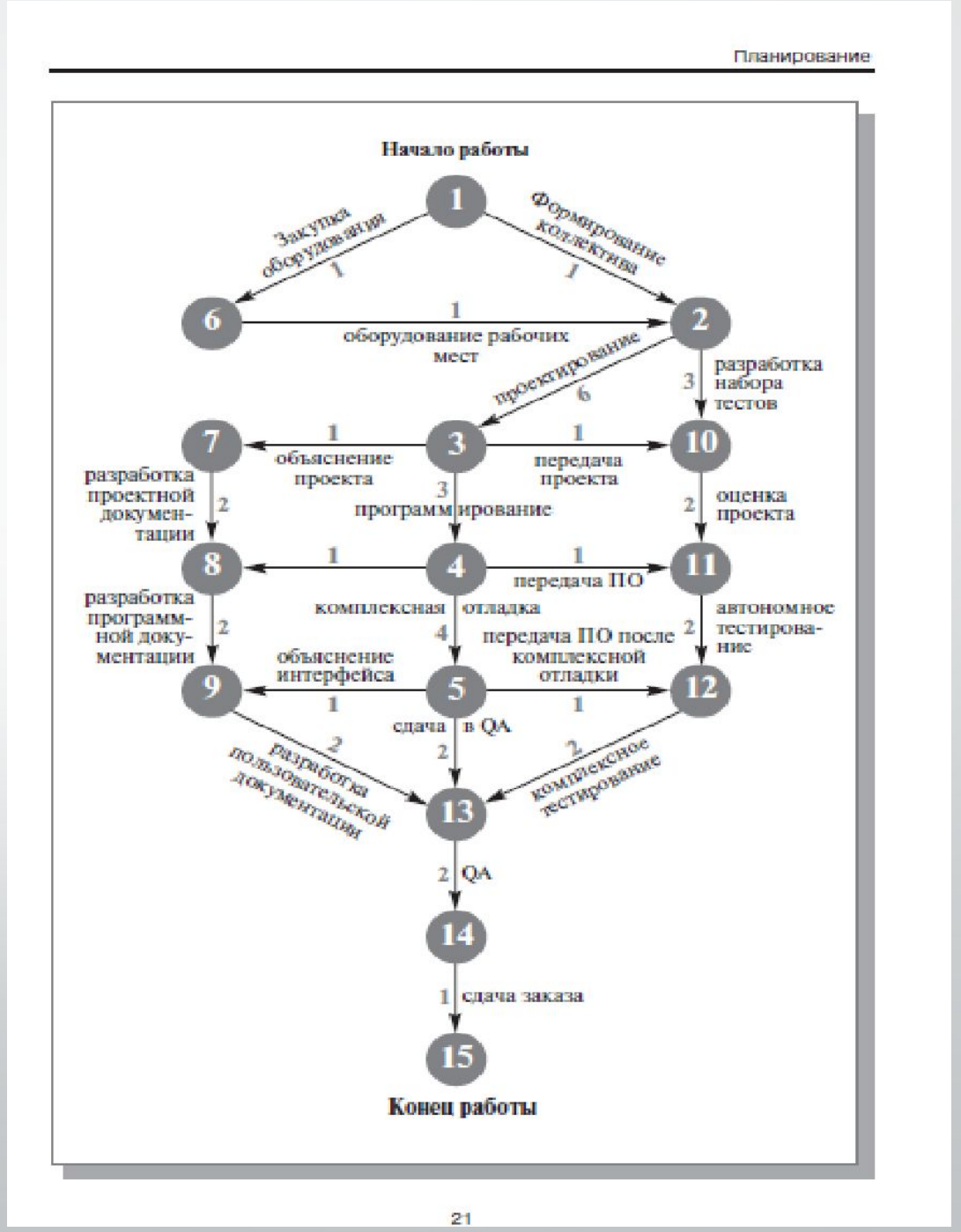
- В настоящее время, когда мы говорим ≪**формализация постановки задачи**≫, мы подразумеваем разработку последовательности моделей, каждая из которых описывает систему и ее окружение с различных точек зрения с постепенной детализацией. Существенно, что все представления о системе, полученные в разных моделях, должны собираться в едином репозитарии (некоторой специальным образом устроенной базе данных) с тем чтобы иметь возможность сквозного проектирования, при котором каждая последующая модель использует результаты предыдущей и уж никак им не противоречит. Соответственно, и все возможные проверки должны быть сквозными.

4. Планирование

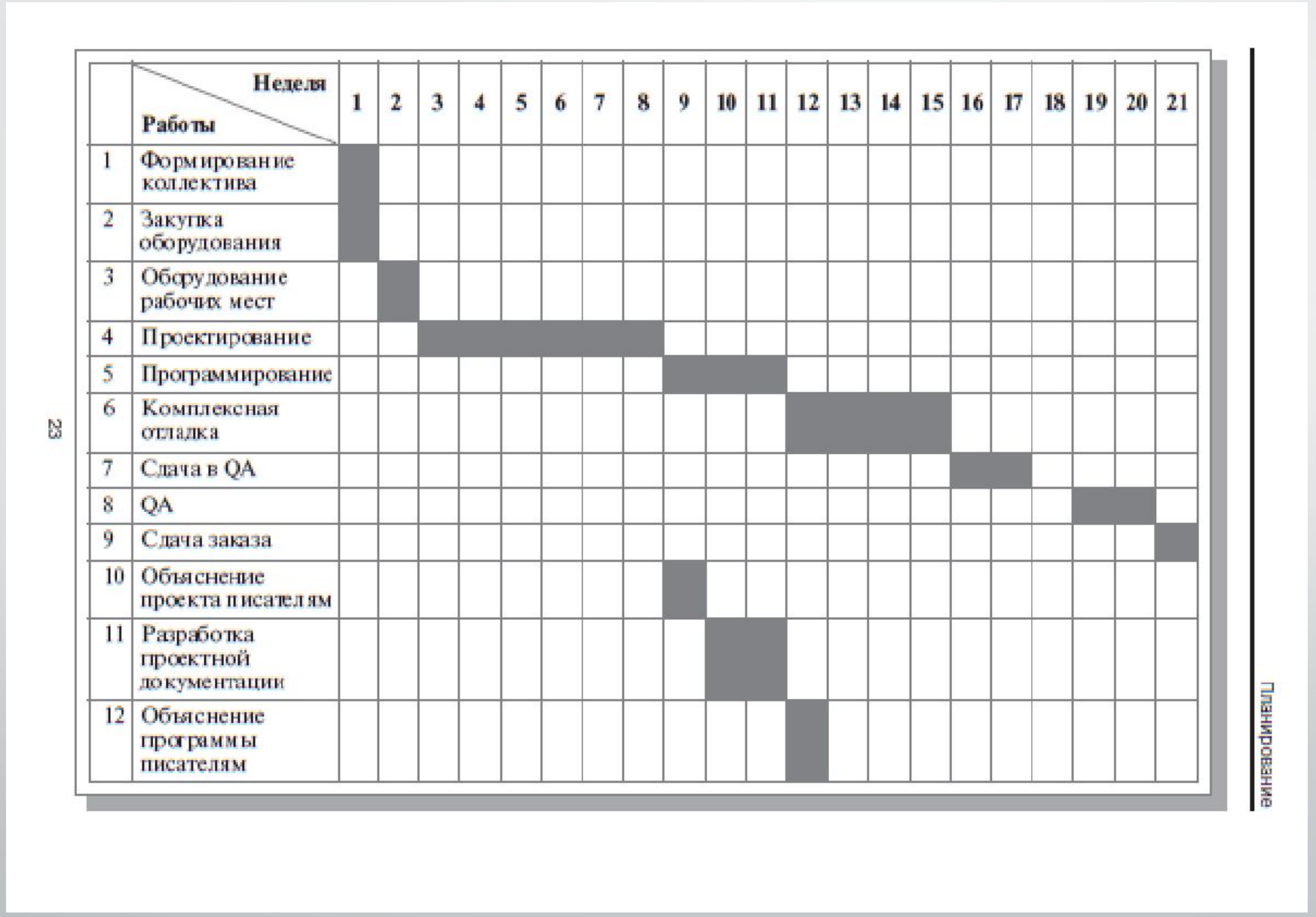
- Результатом фазы оценки осуществимости являются детальная спецификация, план работы и оценка стоимости.
- Наиболее традиционной формой плана можно считать сетевой график, который представляется в виде ориентированного графа с двумя выделенными вершинами – начало и конец работы. Вершинами графа являются события, соответствующие пунктам плана, а ребрами – работы. События должны выражаться глаголами совершенного вида: ≪тексты программ переданы в базу данных исходников≫, ≪все тесты пропущены≫, ≪группа оценки качества дала положительное заключение≫ и т.д., но уж никак не ≪выполняется прогон тестов≫. Ребра нагружаются оценками длительности работ, например, в днях или неделях.
- Сетевой график – очень удобный инструмент: во-первых, на нем четко видна зависимость работ друг от друга, во-вторых, на основе сетевого графика можно вычислить длительность всей работы, в-третьих, пользуясь результатами этих расчетов, можно попытаться оптимизировать длительность и затраты на работу.
- Длительность вычисляется следующим образом. Суммируем длительности работ по всем возможным путям в графе. Тот путь от начала к концу, который является самым длинным, объявляется критическим, потому что задержка любой работы, лежащей на этом пути, приводит к задержке всей работы в целом. Понятно, что критических путей (с одинаковой длительностью) может быть несколько.

4. Планирование

- Рассмотрим пример. Компания получила заказ. Перечислим названия событий, т.е. узлов в графе:
- 1 –начало работы;
- 2 –коллектив сформирован, рабочие места подготовлены;
- 3 –проектирование завершено;
- 4 –программирование завершено;
- 5 –комплексная отладка завершена;
- 6 –оборудование закуплено;
- 7 –группа технических писателей получила описание проекта и необходимые пояснения от проектировщиков;
- 8 –то же для ПО, разработка проектной документации завершена;
- 9 –группа технических писателей получила всю необходимую информацию об интерфейсах с пользователем, разработка программной документации завершена;
- 10 – группа оценки качества (Quality Assurance – QA) разработала тесты;
- 11 – группа QA оценила проект положительно;
- 12 – группа QA завершила автономное тестирование;
- 13 – группа QA завершила комплексное тестирование, получила всю документацию и действующий вариант системы;
- 14 – проверка качества (проблемам качества будет посвящена отдельная лекция) завершена;
- 15 – конец работы (конечно, это не конец, будет еще сопровождение, но пример-то надо закончить).

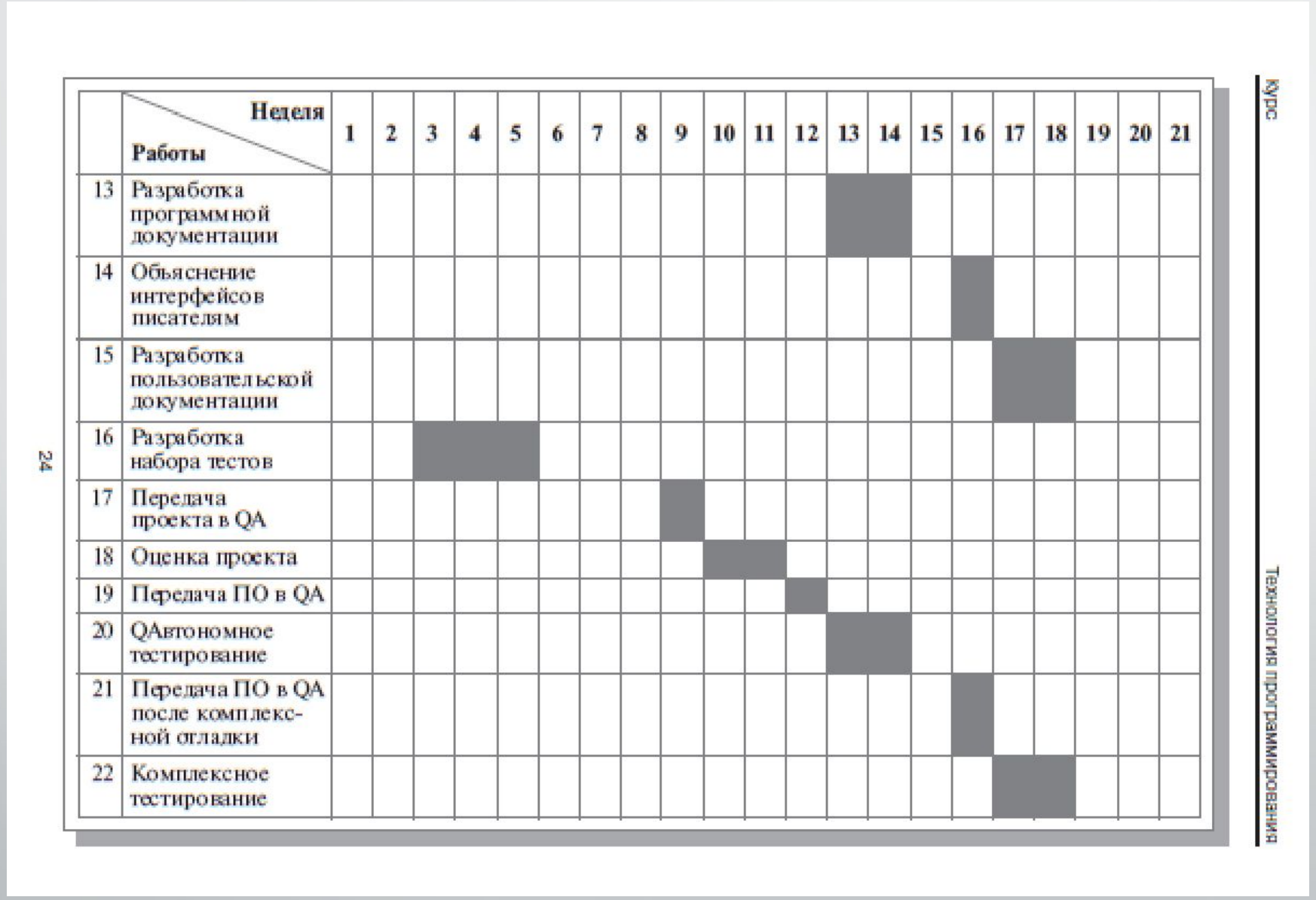


4. Планирование



	Работы \ Неделя	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	Формирование коллектива																					
2	Закупка оборудования																					
3	Обсрудование рабочих мест																					
4	Проектирование																					
5	Программирование																					
6	Комплексная отладка																					
7	Сдача в QA																					
8	QA																					
9	Сдача заказа																					
10	Объяснение проекта писателям																					
11	Разработка проектной документации																					
12	Объяснение программы писателям																					

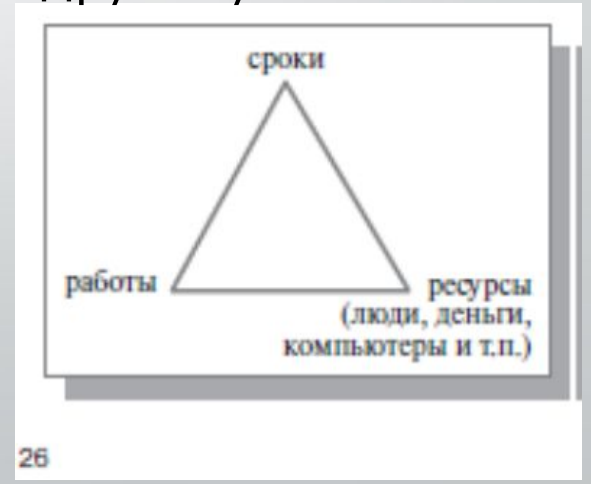
4. Планирование



5. Управление

- Управление почти неотделимо от планирования
- Графически проблему управления можно представить в виде треугольника
- В этом треугольнике каждая вершина зависит от двух других. Например, если работы не укладываются в сроки, можно попытаться договориться с заказчиком об их удлинении (очевидно, что стоимость работ также возрастет). Если сроки сдвинуть нельзя, необходимо договариваться с заказчиками об уменьшении объема работ (т.е. выкинуть какие-то функции из спецификации), либо привлекать дополнительные ресурсы.
- В тех случаях, когда сроки сорваны, ресурсы исчерпаны, а важная работа не выполнена, легче эту работу заново спланировать и поручить другому коллективу.

Опытные руководители проектов отчетливо понимают эти проблемы и решают их, в первую очередь, за счет квалифицированного проектирования, учета возможных рисков, обеспечения возможности регулярного общения между разработчиками, выделения максимально независимых компонентов (которые могут быть переданы дополнительным разработчикам), за счет скрытых ресурсов, о которых разработчики даже не подозревают.



5. Управление

- Однако в качестве рекомендаций или примера можно использовать перечень возможных корректирующих действий менеджера из внутренних правил одной крупной западной фирмы:
- • перераспределите работы, лежащие на критическом пути, так, чтобы они исполнялись более опытными членами коллектива;
- • увеличьте команду исполнителей временными сотрудниками (а не кадровыми);
- • перераспределите исполнителей в нескольких командах (не только в ≪горящей≫);
- • упростите требования к работе;
- • не отвлекайте команду, постарайтесь не прерывать их работу;
- • организуйте дополнительное техническое обучение;
- • если возможно, используйте средства автоматизации разработки;
- • организуйте сверхурочную работу, возможно, многосменную;
- • перепланируйте всю работу, уменьшив число работ на критическом пути (особенно проверьте зависимости одних работ от других).

- Все на уровне здравого смысла, никакой ≪серебряной пули≫, но здесь следует привести фразу: ≪На свете есть множество общеизвестных, но не общепринятых истин≫.
- Если не получается руководить программным проектом, то почему бы не попробовать работать